```
// ASECodeSnippet.txt
// Copyright (c) 2003. Sybase, Inc. All Rights Reserved.
Code snippets showing the algorithm for reading an ASE log:
1. The run() method from RASTranReader.java shows the high level processing.
 public void run()
 {
       synchronized (this)
       {
          _threadStatus = RUNNING;
          notifyAll();
       }
    //  Step 1 identify the location of the mirrored log files
       initializeTranReader();
    // Step 2: reposition ( ie identify the starting point in the log )
       reposition();
    // Steps 3-6 process all of the rows on a page, traverses to the next page,
    // handles end of log, and queues operation for distribution.
       while ( getThreadStatus() != STOP_REQUESTED )
       {
          try
          {
             // Open a log reader session with the log context.
             _dsLogReader = (ASELogReader)_logContext.openSession();
             // Setup the log scan properties.
             LOG_SCAN_PROPS.setScanStartLocator( _currentLocator );
             LOG_SCAN_PROPS.setBlockIncrement( super.getScanSleepIncrement() * 1000 );
             LOG_SCAN_PROPS.setBlockTimeout( super.getScanSleepMax() * 1000 );
             // Configure the scan to return the first log record if
             // the reposition has not yet had been scanned.  Otherwise
             // start scanning from the last known locator, which has
             // really been replicated already.
             if ( _currentLocator.equals( _repositionPoint ) )
                LOG_SCAN_PROPS.setSkipFirstRecord( false );
             else
                LOG_SCAN_PROPS.setSkipFirstRecord( true );
             // Make sure the log scan has the current properties.
             _dsLogReader.setScanProperties( LOG_SCAN_PROPS );
             // Start the scan. This may block waiting for notification
             // that the scan thread has actually started.
             _dsLogReader.startScan();
             ASELogOperation lastOp = null;
             while ( getThreadStatus() != STOP_REQUESTED &&
                   _dsLogReader.hasNext() )
             {
                // Get the next operation from the log reader.
                lastOp = (ASELogOperation)_dsLogReader.nextOperation();
                if ( lastOp == null )
                   continue;
                // Add the operation to queue for distribution
                _operationQueue.put( lastOp );
```

```
            }
            //update the current locator with the scan locator.
            // (or the lastOp locator?)
            _currentLocator = (ASELocator)_dsLogReader.getLocator();
            // Stop the scan.
            _dsLogReader.stopScan();
            sleepToNextScan();
         }
         finally
         {
            synchronized (this)
            {
               if ( _dsLogReader != null )
                  _logContext.closeSession( _dsLogReader );
               _dsLogReader = null;
            }
         }
      }
```

2. Steps 3-6 detail. The scanForward() method shows more detailed log scanning and page traversal algorithm for ASE. It is called by the thread started by ILogReader.startScan() above.

```
 /**
  * scan the log in a forward direction.
  */
 protected void scanForward() throws Exception
 {
//Step 2 set by RASTranReader
    _startRID = ( _scanProperties.getScanStartLocator() instanceof ASELocator ?
                  ((ASELocator)_scanProperties.getScanStartLocator()).getLogOperationRowID() :
                  ((RowID)_scanProperties.getScanStartLocator()) );
    _currentRID = _startRID;
    _nextRID = _currentRID;
    try
    {
       //
       // loop while the scan max operations property hasn't been reached and a stop
       // has not been requested.
       //
       IOperation op = null;
       while ( ! _stopRequested && _operationsBuffered < _scanProperties.getMaxOperations() )
       {
          try
          {
             // read the operation at the nextRID.  May throw InvalidRowIDException
             // if the nextRID points to an invalid page or row number.
             op = (IOperation)readNextOperation();
             // buffer the operation -- must skip nulls
             bufferOperation( op );
          }
          catch (InvalidRowIDException e)
```

```
        {
            //
            // InvalidRowIDException may indicate that the scan has reached the
            // end of a page, or that the scan has hit the end of
            // the active portion of the log.
            //
// Step 5 Obtain next page id
            // need to access the actual page to find if we've reached the
            // end of the log.
            Page p = _logReader.getLogicalPage(_nextRID.getPageNumber());
            int nlp = p.getNextLogicalPageNumber();
            // check end of log condition (next page id is zero)
            if ( nlp > 0 )
            {
               // otherwise create a RowID for the first row on the next page.
               _nextRID = new RowID( nlp, (short)0 );
               continue;
            }
            else if ( nlp == 0 )
            {
// Step 6 Polling on end of log
               // wait for timeout period before checking if page is updated
               block(_nextRID);
               // check if a stop was requested while blocking
               if ( _stopRequested )
                  break;
               // refresh the page from disk and check if the
               // new record is available on the current page
               Page np = null;
               while ( np == null && ! _stopRequested )
               {
                  if ( _logReader.pageModified(p) )
                     np = _logReader.getLogicalPage( _nextRID.getPageNumber() );
                  else
                     block(_nextRID);
               }
               if ( _stopRequested )
                  break;
               assert np != null : "Invalid (null) page refreshed from log.";
               if ( np == null )
                  throw new IllegalStateException("Invalid (null) page refreshed from log.");
               // check if the nextRID row number now exists on the page.
               if ( np.getNextFreeRowNumber() > _nextRID.getRowNumber() )
               {
                  // then the row exists on the newly refreshed page.
                  // just continue to re-read the current value of nextRID
                  continue;
               }
               else if ( np.getNextLogicalPageNumber() != 0 )
               {
```

```
                    // if we now have a next logical page number, then
                    // create a new RowID for the first row on the
                    // next page.
                    _nextRID = new RowID(np.getNextLogicalPageNumber(), (short)0);
                    continue;
                }
                else
                {
                    // the nextRID is not available after blocking on the
                    // end of the log, so throw an EndOfLogException
                    throw new EndOfLogException( p.getLogicalPageNumber() );
                }
            }
        }
    }
    catch (EndOfLogException eole)
    {
// Step 6 reached the end of the log.  (thrown from block() ).  Exit scan
        if ( ASELogReader._logger.isLoggable(Level.FINER) )
            ASELogReader._logger.log(Level.FINER, "Reached end of active portion of the log at Row ID=" + _nextRID);
    }
    catch (Throwable e)
    {
        //
        throw new Exception( "Fatal exception occurred in log scan thread at RowID=" + _nextRID, e );
    }
    finally
    {
        _isBlocked = false;
        synchronized ( _scanBuffer )
        {
            _scanBuffer.notifyAll();
        }
    }
 }
// Operation Builder has reference to DatabaseContext, through which it obtains
// schema information relevant to raw log record.
  /**
   * Reads the next complete log operation from the log, based on the current setting
   * _nextRID.  This takes the next LogRecord object from the LogReader and runs it
   * through the operation builder to build a complete log operation.  The operation
   * builder adds contextual information, such as the schema.  Operation build may
   * read additional raw log record to build the complete operation.
   *
   * @param op_row_id
   * @return
   *
   * @see #readNextLogRecord(LogRecord)
   * @see #processLogRecord(LogRecord)
```

```
     *
     * @throws InvalidRowIDException which percolates up from _logReader.getLogRecord(RowID)
     */
    protected IOperation readNextOperation() throws Exception
    {
        ASELogOperation op = null;
        do
        {
            op = _operationBuilder.buildOperation( readNextLogRecord() );
        } while ( op == null && ! _stopRequested );
        // set the last read locator based on the operation locator.
        if ( ! _stopRequested )
            _currentOperationLocator = (ASELocator)op.getLocator();
        return op;
    }
    /**
     * Reads the next raw LogRecord from the log based on the current setting of
     * _nextRID.
     *
     * @return
     */
    protected LogRecord readNextLogRecord() throws Exception
    {
        LogRecord lr = null;
        do
        {
// Read raw log record from disk          // get the raw operation from the log reader.
            lr = _logReader.getLogRecord( _nextRID );
            // update current/next row ids
            _currentRID = _nextRID;
            _nextRID = _nextRID.getNextRowID();
            // increment total ops read statistics
            _totalOperationsRead++;
            // if this is the first operation read and the scan properties request
            // skipping the first record, drop it and read the next one
        } while ( _totalOperationsRead == 1 &&
                  _scanProperties.isSkipFirstRecord() &&
                  ! _stopRequested );
        return lr;
    }
```

3. Detail from ASELogReader.java showing obtaining a page from underlying mirrored device.

```
    /**
     * @param rowid
     * @return
     */
    protected LogRecord getLogRecord(RowID rowid) throws Exception, InvalidRowIDException
    {
        // get the specific virtual page
        Page p = getLogicalPage( rowid.getPageNumber() );
        // get the row buffer from the page.
```

```
    ByteBuffer buffer = p.getRowBuffer(rowid.getRowNumber());
    // if the row is not found on the page (page returns null), then throw an exception.
    if ( buffer == null )
       throw new InvalidRowIDException( rowid );
    // create and return a log record for the current row.
    return LogRecordFactory.createLogRecord(p, buffer);
 }
     /**
  * Get the page referenced by the logical page number.
  *
  * @param pageBuffer a ByteBuffer in which to place the data read from the device
  * @param lpage the logical page id to read.
  *
  * @return Page object encapsulating the read buffer.
  *
  * @throws Exception
  */
 synchronized protected Page getLogicalPage(int lpage) throws Exception
 {
    // if the requested page is the currently held current page, return that.
    // Note: the pageModified() check may be expensive!
    if ( _currentPage != null &&
        _currentPage.getLogicalPageNumber() == lpage &&
        ! pageModified( _currentPage ) )
       return _currentPage;
    // get the current device based on the logical page id.
    IASEDeviceContext dev = _databaseContext.getDeviceForLogicalPage( lpage );
    // open the device if it is not already.
    if ( ! dev.isOpen() )
       dev.open();
    // calculate the virtual page number from the logical page id.
    // may throw an exception if the corresponding device map can't be found.
    long vpage = _databaseContext.getVirtualPageFromLogicalPage(lpage);
    // obtain a page buffer to use for the page
    ByteBuffer pageBuffer = getPageBuffer();
    // read the virtual page into the page buffer.
    dev.readVirtualPage(pageBuffer, vpage );
    // rewind the buffer to prepare it for reading.
    pageBuffer.rewind();
    // create and return a new Page constructed from the page buffer.
    Page p =  PageType.createPage( pageBuffer, (int)_databaseContext.getLogicalPageSize() );
    return _currentPage = p;
 }
// OracleCodeSnippet.txt
// Copyright (c) 2003. Sybase, Inc. All Rights Reserved.
Code snippets showing the algorithm for reading an Oracle log:
Since the Oracle log is made up of many files, the nextRecord() method in OracleLogReader abstracts
individual log files to make them appear as one indvidual log file.  The OracleLogReader has a
reference to an OracleLogFileReader (_currentLogFile).  The OracleLogFileReader represents
an individual Oracle log file.
```

1. The run() method from RAOTranReader.java shows the high level processing.

```
public void run()
{
    Object scanResults = null;
    int numOps = 0;
    try
    {
  // Step 1 Identify the location of the mirrored log file(s)
      startReplication();
// Step 2 Identify the starting log id, block id and record offset of the active portion of the log with respect to replication.
// Step 3 Open the log specified by the log id from disk.
      // Position in the transaction log before beginning first log scan
      position();
      while ( !(_stopImmediate || _stopQuiesce) )
      {
        try
        {
          // Execute the log scan and get results back.
          // If no data is found, a NoDataAvailException is thrown.
          scanResults = executeLogScan();
          // Process log operations.
          numOps = processData(scanResults);
        }
        catch (NoDataAvailException ndae)
        {
          // sleep based on configured increment and max
          sleepToNextScan();
        }
      }
    }
    catch (TranReaderStoppedException ex)
    {
      // Received stop request ... just move on to the finally block.
    }
    catch (TranReaderException ex)
    {
      // Fire the appropriate event.
      Throwable t = ex.getCause();
      if (  t != null && t instanceof ConnException )
      {
        _threadError = t;
        LRThreadEvent.fireEvent( _connectionMgr, (ConnException)t, this );
      }
      else
      {
        _threadError = ex;
        // This gets the TRConnectFailureException.
        LRThreadEvent.fireEvent(ex);
      }
    }
```

```
        catch (Throwable t)
        {
            _threadError = t;
            // Catch all other possible exceptions and treat them as fatal errors.
            LRThreadEvent.fireEvent( LRThreadEvent.THREAD_ERROR_FATAL, this );
        }
        finally
        {
            _threadStatus = ( _threadError == null ? STOP_NORMAL : STOP_ERROR );
        }
    }
```

2. The position() method from RAOTranReader.java and the open() method from OracleLogReader show the opening and positioning of the OracleLogReader for reading.

```
// Step 2 Identify the starting log id, block id and record offset of the active portion of the log with respect to replication.
// Step 3 Open the log specified by the log id from disk.
    private synchronized void position() throws Exception
    {
        if (_stopImmediate || _stopQuiesce)
            throw new TranReaderStoppedException();
        _olr.open( ((RAOLocator)_truncPoint).getLogLocator() );
    }
    /**
     * Opens the OracleLogReader for reading at the location in the log specified
     * by the locator.
     *
     * @param locator specifies the position in the log to start reading.
     * @throws Exception if an error occurs.
     */
    public void open(ILocator locator) throws Exception
    {
// Log ID, block ID, offset
        try
        {
            LogLocator loc = (LogLocator)locator;
            Number logID = loc.getOldestActiveTranLogID();
            Number blockID = loc.getOldestActiveTranBlockID();
            Number blockOffset = loc.getOldestActiveTranBlockOffset();
            if (logID.intValue() == 0)
            {
                logID = loc.getLogID();
                blockID = loc.getBlockID();
                blockOffset = loc.getBlockOffset();
                if (logID.intValue() == 0)
                    this.open();
                else
                {
                    if (blockID.intValue() != 0 && blockOffset.intValue() != 0)
                        this.open(logID, blockID, blockOffset);
                    else if (blockID.intValue() != 0)
                        this.open(logID, blockID);
```

```
            else
               this.open(logID);
         }
      }
      else
      {
         this.open(logID, blockID, blockOffset);
      }
   }
   catch(Exception ex)
   {
      throw new OracleLogReaderException("OLR_OPENERR", ex);
   }
}
```

3. The nextBlock() method from LogFileReader.java reads the next block from the Oracle log file.

```
/**
 * Loads the next block in the log.
 *
 * @return the next block in the log.
 * @throws Exception
 */
private void nextBlock() throws Exception
{
   // Get the next block from the file
   _currentBlock = ByteBuffer.allocate(BLOCK_SIZE);
// Step 4 Read the next block into memory.
   _file.read(block);
   // Construct the block header from this block.
   _currentBlockHeader = new BlockHeader(block);
   // If the log id found in the new block is different than the expected log id,
   // the end of the log has been reached.
// Step 5 Check that the log id of the block just read is equal to the specified log id.
   if(!_currentBlockHeader.getLogID().equals(_logID))
      throw new EndOfLogException();
   return block;
}
```

3. The nextRecord() method from LogFileReader.java reads the next record from the Oracle log file.

```
/**
 * Return the next log record in a log file.
 *
 * @return the next log record in a log file.
 * @throws Exception
 */
private LogRecord nextRecord() throws Exception
{
   if(_currentBlock == null)
   {
      // On startup or when the end of the log has been reached, get the next block.
```

```
            nextBlock();
            int offset = _currentBlockHeader.getRecordOffset().intValue();
            // If the offset is 0, move to the next block.
            while(offset == 0)
            {
               nextBlock();
               offset = _currentBlockHeader.getRecordOffset().intValue();
            }
            // Position the block at the offset of the first record.
            _currentBlock.position(offset);
         }
         else if(_currentBlock.position() == BLOCK_SIZE)
         {
            // If the current position is at the end of the block, get the next block.
            nextBlock();
         }
         // If the record header for the next record can't fit on this block, then the rest
         // of the data on this block is garbage.  Just move to the next block for the
         // the next record.
         int position = _currentBlock.position();
         if(position >= (this.getBlockSize() - 8))
            nextBlock();
// Step 6 Read the number of bytes corresponding to the length of a record header and construct an object to
represent it.
         RecordHeader recordHeader = new RecordHeader(_currentBlockHeader, _currentBlock);
// Step 7 Based on the length value in the record header:
//       a. If the length is zero, repeat step 4 - 7
//       b. If the length is non-zero, read the number of bytes equal to length.  Repeat steps 6 - 7.
         int recordLength = recordHeader.getRecordLength().intValue();
         if(recordLength == 0)
         {
            // Go to the next block.
            nextBlock();
            // Recursively return the next record.
            return nextRecord();
         }
         else
         {
            // Allocate a buffer for the record body.
            recordBody = ByteBuffer.allocate(recordLength - RecordHeader.LENGTH);
            while(recordBody.hasRemaining())
            {
                     // Get the next byte.
                     recordBody.put(this.currentBlock().get());
            }
         }
//
         return LogRecordFactory.createRecord(recordHeader, recordBody);
      }
```

5. The nextRecord() method from OracleLogReader.java abstracts individual log files

to make them appear as one indvidual log file.

```
   /**
    * Return the next record in the log.
    *
    * @return the next record in the log.
    * @throws Exception
    */
   public LogRecord nextRecord() throws Exception
   {
      LogRecord record = null;
      try
      {
         // Save the current block id and the current block offset in case this is the very end of the log.
         _currentBlockID = _currentLogFile.currentBlockID();
         _currentBlockOffset = _currentLogFile.currentBlockOffset();
         // Get the next record.
         record = (LogRecord)_currentLogFile.nextRecord();
      }
      catch(EndOfLogException ex)
      {
         // Is this the end of the current log file or the very end of the log?
         // If there is a more recent log that was written, then it's just the end of the current log.
// Step 5 Check that the log id of the block just read is equal to the specified log id.
//       If the log id of the block is greater than the specified log id, the end of the log has been reached, so re-position to
read the block again and repeat steps 4 - 5.
//       If the log id of the block is less than the specified log id, the log has been overwritten by a newer log, so close
the log and repeat steps 3 - 5.
         Number latestSQN = _logMD.getCurrentLog().getLogID();
         Number currentSQN = _currentLogFile.getLogID();
         if(currentSQN.equals(latestSQN))
         {
            // This is the very end of the log.
            // Reposition for the next read before throwing the exception.
            _currentLogFile.position(_currentBlockID, _currentBlockOffset);
            throw ex;
         }
         else
         {
            // There is a new log file to read.
            // Open the new log file.
            LogFile newLog = null;
            try
            {
               // Get the next log sequence number.
               Number nextSQN = _logMD.getNextLogSequence(currentSQN);
               boolean open = false;
               while(!open)
               {
                  try
                  {
```

```
                // Get the log file with the given sequence number.
                newLog = _logMD.getLog(nextSQN);
                // Open the log file.
                newLog.open();
                open = true;
            }
            catch(EndOfLogException ex2)
            {
                // The possibility exists that the new log has not been written to yet.
          // If the the new log is still the latest log to be written, just throw an exception.
                if(newLog.getLogID().equals(latestSQN))
                   throw ex2;
                // Otherwise, the log is empty, so move to the next log file.
                newLog = _logMD.getLog(_logMD.getNextLogSequence(nextSQN));
                newLog.open();
                open = true;
            }
          }
        }
        catch(Exception e)
        {
    // A severe error has occured.
          // Try to close the new log.
          try { newLog.close(); }
          catch(Exception ignore) {}
          // Try to re-position for reading in the current log.
          _currentLogFile.position(_currentBlockID);
          _currentLogFile.position(_currentBlockOffset);
          throw e;
        }
        // The log switch was successful, so close the previous log before setting the current log to the new log.
        _currentLogFile.close();
        _currentLogFile = newLog;
        // Recursively get the next record from the new log.
        record = nextRecord();
    }
}
catch(InconsistentLogException ex)
{
    // The possibility exists that, if we are reading an online log, the
    // current log could be overwritten as we are reading it.  This just
    // means we need to load the archived version of this log and start
    // reading from where we left off.
    // Get the current log's sequence number and save the current position.
    Number sqn = _currentLogFile.getLogID();
    _currentBlockID = _currentLogFile.currentBlockID();
    _currentBlockOffset = _currentLogFile.currentBlockOffset();
    // Close the online log.
    _currentLogFile.close();
    // Get and open the archived log.
```

```
        _currentLogFile = _logMD.getLog(sqn);
        _currentLogFile.open();
        // Re-position to where we left off.
        _currentLogFile.position(_currentBlockID);
        _currentLogFile.position(_currentBlockOffset);
        // Get the next record.
        record = (LogRecord)this.nextRecord();
    }
    catch(UnsupportedRecordException ex)
    {
        throw ex;
    }
    return LogRecordFactory.createRecord(record);
}
```